RENDERING STOCHASTIC & ACCUMULATION BUFFER UNTUK EFEK MOTION BLUR PADA ENGINE OGRE 3D

Richard Pramono
Universitas Surabaya
richardpramono@gmail.com

Abstrak. Sebuah foto dari obyek yang bergerak dengan cepat akan menghasilkan efek *motion blur*. Sebaliknya, seluruh hasil proses *render* komputer grafis akan menghasilkan gambar yang tajam. Untuk menghasilkan hasil proses *render* yang realis, dibutuhkan efek *motion blur*. Banyak pendekatan dilakukan untuk menghasilkan efek *motion blur*, antara lain *accumulation buffer*, *post-process motion blur*, dan metode *stochastic*. Dalam jurnal ini, kami mengembangkan serta membandingkan *motion blur* pada engine OGRE 3D. Metode yang digunakan adalah *accumulation buffer* dan *metode stochastic*. Dibandingkan dengan metode *accumulation buffer*, metode *stochastic* dapat mengurangi artifak bergaris yang dihasilkan metode *accumulation buffer*. Namun metode *stochastic* dapat menghasilkan *noise* acak.

Kata Kunci: Motion blur, stochastic rendering, accumulation butter, OGRE 3D engine.

Kamera menangkap cahaya dengan membuka shutter dalam interval waktu tertentu untuk menerima cahaya. Semakin lama interval, semakin banyak intensitas cahaya yang diterima. Apabila obyek bergerak saat *shutter* kamera terbuka, gambar yang dihasilkan akan memiliki motion blur. Grafika komputer menggunnakan *shutter instant*[1]. Hal ini menyebabkan grafika komputer tidak dapat memberikan efek *motion blur*.

Karena sistem visual manusia juga menghasilkan *motion blur*, sebaiknya gambar grafika komputer perlu untuk diberi efek *motion blur*. Contoh aplikasi yang dapat diberi efek *motion blur* antar lain animasi dan aplikasi interaktif seperti game, *virtual reality* dan *augmented reality*[2].

OGRE (Object-Oriented Graphic Rendering Engine) adalah *engine render* 3D yang ditulis dalam bahasa C++. Penelitian ini akan menggunakan *engine* OGRE untuk mengimplementasikan algoritma *motion blur*.

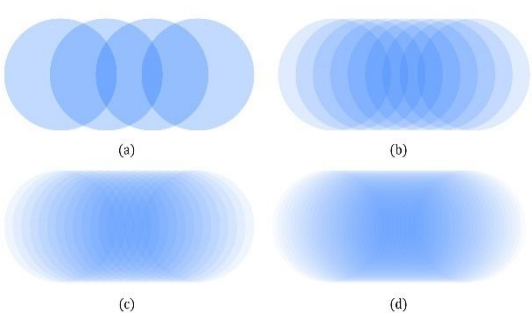
Dikarenakan *engine* OGRE gratis dan *open-source*, sehingga amat memungkinkan untuk dilakukan pengembangan ke depan.

I. METODOLOGI

Pendekatan *motion blur* dalam grafika komputer telah dilakukan sejak 1983. Hingga kini telah banyak algoritma yang dikembangkan untuk mendapatkan efek *motion blur*. Secara umum, terdapat tiga buah pendekatan efek *motion blur* yang umum digunakan, antara lain (1) menggunakan *accumulation buffer*, (2) *post-process motion blur*, dan (3) *stochastic rasterization motion blur*. Selain tiga jenis di atas, ada algoritma khusus *motion blur* untuk beberapa tujuan spesifik, seperti *motion blur* untuk *scanline renderer*[3], *rendering* berbasis titik[4], *ray tracing* [5] dan bahkan *motion blur* untuk simulasi cairan [6]. Namun hanya tiga *motion blur* yang dapat diaplikasikan secara *real-time* yang akan dibahas.

Motion Blur menggunakan *Accumulation Buffer*

Dengan melakukan *render* lebih dari sekali (***N***) setiap selisih waktu (Δt), maka gambar akan terlihat kabur (apabila objek bergerak) seperti pada Gambar 1. Namun apabila jumlah render terlalu sedikit, maka akan tampak efek bergaris. Dengan mengunakan metode ini, pergerakan kamera/objek harus cukup kecil secara relatif terhadap interval yang diambil.



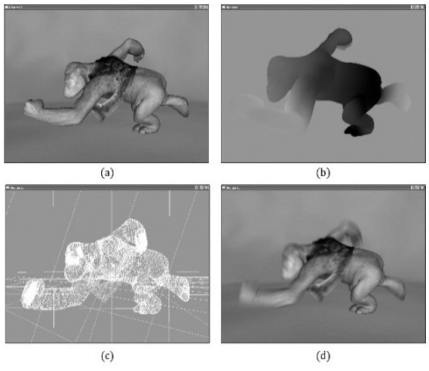
Gambar 1. *Accumulation buffer*

Kelebihan dari *accumulation buffer*, adalah *visibility* dari urutan objek pasti benar untuk setiap waktu (***N*** kali render). Namun untuk pemrosesan geometri, algoritma ini sangat mahal. Karena (1) scene perlu di-*render* ***N*** kali hanya untuk menampilkan satu buah gambar dalam setiap Δt, (2) dan untuk setiap objek yang tidak bergerak dalam periode Δt juga terpaksa di-*render* juga, ini menyebabkan *accumulation buffer* tidak efisien. Namun objek yang tidak bergerak juga wajib diperhitungkan, karena dapat mempengaruhi hasil *motion blur*. Karena *motion blur* dapat terjadi di depan ataupun di belakang objek yang tidak bergerak.

Dalam menggunakan *accumulation buffer*, pembagian jumlah waktu ***N*** wajib diperhitungkan. Karena bila terlalu sedikit, maka akan muncul artifak bergaris [7][8]. Apabila pergerakan lambat, maka *accumulation buffer* akan tidak efisien. Jumlah ***N*** idealnya secara relatif berbanding lurus dengan kecepatan gerak objek dibandingkan posisi kamera. Pada umumnya *motion blur* menggunakan *accumulation buffer* tidak dapat dilakukan secara *real time* -namun sekarang telah memungkinkan- dengan catatan *scene* tidak terlalu kompleks.

Motion Blur sebagai Post-Prosess

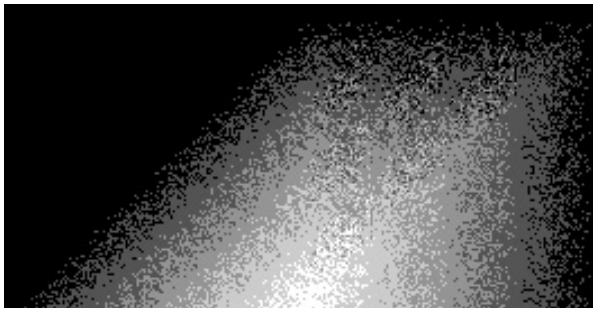
Pada 2003, Simon Green mengimplementasikan *motion blur* sebagai efek *post-process* [9]. Efek post-process artinya efek *motion blur* yang dihasilkan dihitung setelah gambar di-*render*. Ini bukan *motion blur* sebenarnya, namun hasilnya sudah tampak cukup baik. Simon membuat *buffer* kecepatan dengan menghitung perubahan posisi vertex sebelum dan setelah Δt untuk setiap periode. Contoh *motion blur* sebagai post-process dapat dilihat pada Gambar 2.



Gambar 2. *Post-Processing Motion Blur*

Pendekatan *Motion Blur Stochastic*

Dalam dunia nyata, *motion blur* yang dihasilkan foto akan memiliki *noise*. Ini dikarenakan kecepatan *shutter* yang rendah cenderung menghasilkan *noise* pada sensor atau film kamera. *Stochastic rasterization* telah dikembangkan sejak 1986 untuk menghasilkan *penumbra*, pantulan yang kabur, *depth of field* (Dof) dan *motion blur* [10]. Stochastic motion blur yang dihasilkan secara *real-time* diciptakan pada tahun 2007. Ini dapat dilihat di Gambar 3. Namun saat itu masih tidak dapat diimplementasikan menggunakan GPU konvensional [11]. Pendekatan *stochastic* pada umumnya dilakukan menggunakan kerangka *ray tracing*, sedangkan struktur kartu grafis saat ini didesain untuk melakukan *Forward Rendering*. Pendekatan *stochastic real-time* pada GPU konvensional dilakukan oleh McGuire tahun 2010.



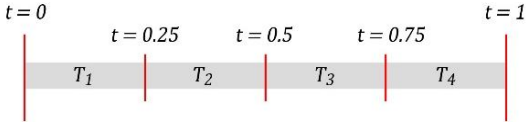
Gambar 3. *Stochastic Motion Blur*

Noise yang dihasilkan oleh *stochastic rasterization* dikarenakan sampel yang dipilih tidak terletak pada satu perbedaan waktu yang sama. Sehingga pixel yang dihasilkan akan berbeda satu dengan yang lain. Pengacakan ini dapat mengurangi efek bergaris pada *accumulation buffer*. Sehingga jumlah *sampling* yang diambil dapat dikurangi, umumnya Multi Sampling Anti Alias (MSAA) atau Super Sampling Anti Alias (SSAA) digunakan untuk mencampur *noise* dengan pixel yang berdekatan, meningkatkan kualitas hasil.

Algoritma yang akan digunakan adalah melakukan penggabungan pendekatan stochastic dan accumulation buffer pada OGRE. Sehingga ada banyak batasan yang terpaksa dilakukan, karena secara normal OGRE tidak dapat melakukan *rendering* dengan kerangka *ray tracing*. Untuk dapat mengambil sampel yang cukup untuk *stochastic*, maka digunakan *accumulation buffer*. Sampel ini akan disimpan pada sebuah tekstur menggunakan fitur Render To Texture (RTT) milik OGRE.

Stochastic Rasterization untuk *Motion Blur*

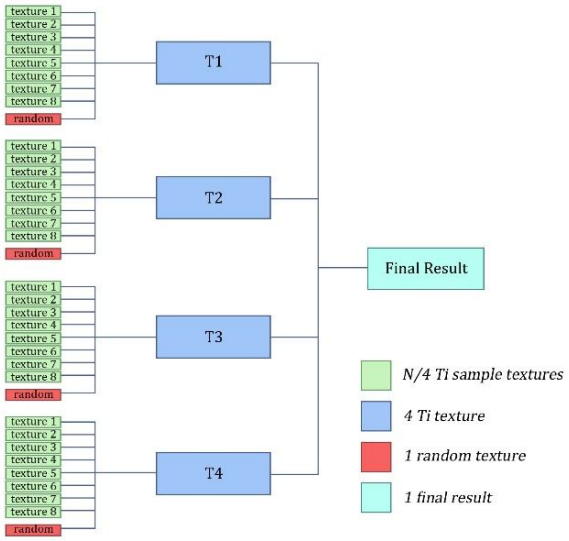
Dikarenakan objek bergerak, maka kita dapat mengasumsikan waktu scene ditulis dalam dua interval Δt yang telah dinormalisasi, t=0 hingga t=1 seperti pada Gambar 4. Di mana t=0 adalah waktu gambar sebelumnya dan t=1 adalah waktu gambar sesudahnya. Waktu interval t dibagi menjadi empat, yakni T1, T2, T3 dan T4. Dimana T_i dalam interval $\left[\frac{i}{n}, \frac{i+1}{n}\right)$ dan i ∈ {0, … , n-1 }. Untuk setiap T, diambil sampel t_i dari *accumulation buffer*.



Gambar 4. Pembagian sampel T1-T4

Accumulation buffer

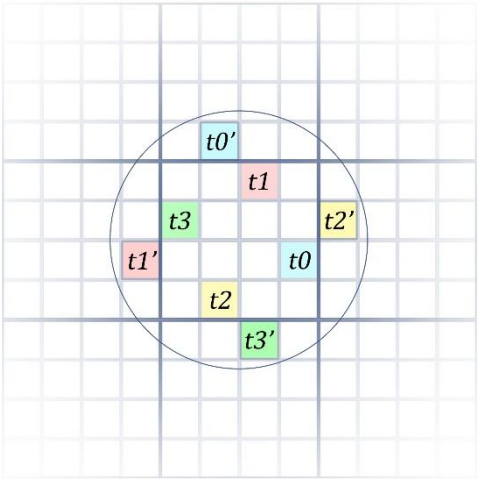
Accumulation buffer dipakai untuk menyediakan sampel T1, T2, T3 dan T4. Empat tekstur dibutuhkan untuk menyimpan masing-masing T. Karena GPU tidak dapat menghitung fungsi acak, maka nilai acak dihitung di CPU kemudian dikirim ke GPU menggunakan sebuah *buffer* tekstur. Channel R, G, B dan A dipakai untuk menyimpan nilai random. Sehingga didapat 20 tekstur (5 RTT * 4 T) yang perlu dikirim ke fragment shader setiap frame. Tapi karena tekstur acak sudah dihitung sebelumnya, cukup 16 tekstur yang dibutuhkan untuk setiap *frame*. Angka ini tergantung jumlah *accumulation buffer* yang digunakan, dalam contoh ini 16 buffer menghasilkan hasil *render* yang cukup baik. Kerangka *render* ini dapat dilihat pada Gambar 5.



Gambar 5. Struktur *render*

Pengambilan Sampel

Setelah *accumulation buffer* disimpan, MSAA dilakukan untuk mengurangi garis-garis yang tajam. Sampel yang digunakan adalah Rotated Grid Super Sampling (RGSS) yang dimodifikasi seperti tertera pada Gambar 6.



Gambar 6. Sampel RGSS

Sampel RGSS diambil dari 8 posisi, di mana setiap masing-masing sampel diambil dari waktu yang berbeda. Sampel diambil pada t_0, t_1, t_2, t_3, t_0', t_1', t_2' dan t_3'. Sampel t_0 hingga t_3 diambil dari dalam pixel dan sampel t_0' hingga t_3' diambil dari pixel sebelah. Di mana t_0 dan t_0' diambil dari T1, t_1 dan t_1' diambil dari T2, t_2 dan t_2' diambil dari T3 dan t_3 dan t_3' diambil dari T4. Semua nilai t_i diambil dari waktu yang diacak sebelumnya.

Warna hasil kemudian dihitung menggunakan perbedaan berat, dikarenakan seharusnya sampel yang diambil di dalam pixel (t_0-t_3) akan lebih mempengaruhi pixel hasil dibandingkan sampel dari pixel sebelah (t_0'-t_3'). Diasumsikan warna di dalam pixel dinotasikan sebagai c^0_l, di mana $l \in \{0, 1, 2, 3\}$ dan warna dari pixel sebelah dinotasikan sebagai c^1_l di mana $l \in \{0, 1, 2, 3\}$.

$$C = w_0 \sum_{l=0}^{3} c_l^0 + w_1 \sum_{l=0}^{3} c_l^1$$

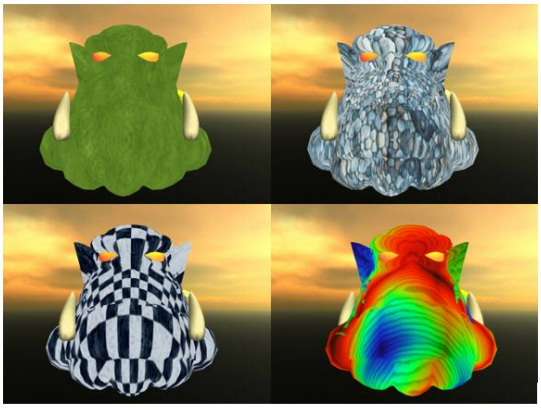
Persamaan 1. Distribusi berat warna dari sampel dalam pixel dan sampel luar pixel

Nilai yang digunakan adalah $w_0 = 5/32$ dan $w_1 = 3/32$, dikarenakan dapat menciptakan hasil *render* yang baik.

II. HASIL DAN PEMBAHASAN

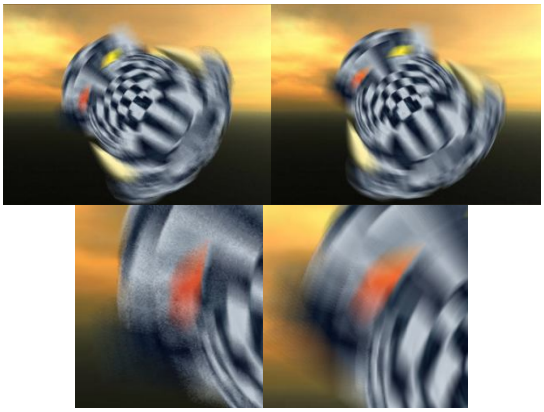
Setelah metode diimplementasikan, dilakukan percobahan terhadap sebuah objek dengan berbagai variasi tekstur dan berbagai variasi gerakan. Variasi tekstur yang dipilih antara lain tekstur dengan kontras rendah, sedang, tinggi dan pelangi. Variasi tekstur tampak pada Gambar 7. Variasi gerakan yang dipilih antara lain rotasi dan translasi. Hasil kemudian diamati dan dicatat waktu rata-rata dari seluruh percobaan. Hasil dibandingkan dengan algoritma *accumulation buffer* standar dengan variasi gerakan, kecepatan dan tekstur yang sama.

Rendering Stochastic tampak lebih natural dan dapat mengurangi efek bergaris *accumulation buffer* secara signifikan. Hal ini tampak pada Gambar 8 dan 9. Namun dari jarak jauh, noise yang dihasilkan bisa



Gambar 7. Empat jenis tekstur yang digunakan

Gambar 8. Perbandingan kualitas *motion blur* translasi kecepatan sedang dengan *stochastic* (kiri) dan *accumulation buffer* (kanan) dengan kontras tekstur rendah



Gambar 9. Perbandingan kualitas *motion blur* rotasi kecepatan tinggi dengan *stochastic* (kiri) dan *accumulation buffer* (kanan) dengan kontras tinggi

tampak mengganggu. Dengan kecepatan translasi dan yang sama, *stochastic* dan *accumulation buffer* memiliki hasil yang tidak jauh berbeda. Namun ketika menggunakan kecepatan yang lebih tinggi, *stochastic rendering* tampak lebih baik dibandingkan *accumulation buffer*. Hal ini dikarenakan kecepatan tinggi akan menghasilkan efek bergaris yang lebih terlihat pada *accumulation buffer*. Pada kedua algoritma, hasil kualitas tekstur tampak baik. Tingkat kontras tekstur tidak mempengaruhi kualitas *motion blur* yang dihasilkan. Kecepatan frame per detik masih *real-time*, berkisar antara 28.4-30.0. Tabel hasil simulasi dapat dilihat di Tabel 1.

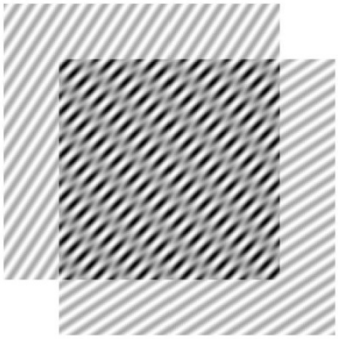
Tabel 1. Hasil rata-rata frame per detik waktu render

		Low Contrast	Medium Contrast	High Contrast	Colorful
Slow Rotation	ACC	29.8	29.7	29.7	28.8
	STO	30.0	29.7	29.8	29.7
Fast Rotation	ACC	29.3	29.7	29.7	29.0
	STO	29.6	29.8	29.5	29.0
Slow Translation	ACC	28.4	29.0	28.1	28.6
	STO	28.9	29.4	26.4	29.4
Fast Translation	ACC	29.5	29.7	29.8	29.2
	STO	29.5	29.6	29.5	28.9

III. SIMPULAN

Stochastic rasterization pada umumnya menggunakan *ray tracing* untuk mengurangi jumlah sampling. Namun karena OGRE tidak dapat menggunakan *ray tracing*, maka *accumulation buffer* digunakan untuk mengambil sampel *stochastic*. Hasil *motion blur stochastic* tampak cukup baik dan menghilangkan kekurangan algoritma

accumulation buffer, yakni efek bergaris dan pola Moiré. Pola Moiré adalah pola yang muncul dikarenakan tingkat kontras tekstur yang tinggi pada gambar digital. Contoh pola Moiré dapat dilihat pada Gambar 10.



Gambar 10. Pola Moiré seperti ini dapat dihilangkan dengan cara mengurangi kontras

Namun karena sampel *stochastic* yang dipakai bukan metode *ray tracing*, noise yang dihasilkan tampak mengganggu pada gambar dengan kontras tinggi. Waktu acak T1, T2, T3 dan T4 tidak murni *stochastic*, tapi *stochastic* berdasar pattern tertentu.

IV. DAFTAR PUSTAKA

[1] McGuire, M., Enderton, E., Shirley, P., Luebke, D. (2010). Real-time Stochastic Rasterization on Conventional GPU Architectures. Proceedings of the Conference on High Performance Graphics, pp. 173-182.

[2] Wloka, M. M., Zeleznik, R. C. (1996). Interactive Real-Time Motion Blur. The Visual Computer Vol. 12, No. 6, pp. 283-295.

[3] Korein, J., Badler, N. (1983). Temporal Anti-Aliasing in Computer Generated Animation. Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 377-388.

[4] Heinzle, S., Wolf, J., Kanamori, Y., Weyrich, T., Nishita, T., Gross, M. (2010). Motion Blur for EWA Surface Splatting. Computer Graphics Forum Vol. 29, No.2, 733-742.

[5] Neilson, D., Yang, Y. H. (2003). Variance Invariant Adaptive Temporal Supersampling for Motion Blurring. Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, pp. 67.

[6] Kim, D., Ko, H. (2007). Eulerian Motion Blur. Proceedings of the Eurographics Workshop of Natural Phenomena, pp. 39-46.

[7] Deering, M., Winner, S., Schediwy, B., Duffy, C., Hunt, N. (1988). The Triangle Processor and Normal Vector Shader: a VLSI System for High Performance Graphics. Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 21-30.

[8] Haeberli, P., Akeley, K. (1990). The Accumulation Buffer: Hardware Support for High-Quality Rendering. Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 309-318.

[9] Green, S. (2003). OpenGL Shader Tricks. Talk at Games Developer Conference.

[10] Cook, R. L. (1986). Stochastic Sampling in Computer Graphics. ACM Transactions on Graphics Vol. 5 No. 1, January 1986 , pp. 51-72.

[11] Akenine-Möller, T., Munkberg, J., Hasselgren, J. (2007). Stochastic Rasterization using Time-Continuous Triangles. Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics Hardware, pp. 7-16.